Eternity Variables to Prove Simulation of Specifications

Hesselink, Wim H.

Published in:
ACM Transactions on Computational Logic

Publication date:
2005

Eternity Variables to Prove Simulation of Specifications

WIM H. HESSELINK
University of Groningen

Simulations of specifications are introduced as a unification and generalization of refinement mappings, history variables, forward simulations, prophecy variables, and backward simulations. A specification implements another specification if and only if there is a simulation from the first one to the second one that satisfies a certain condition. By adding stutterings, the formalism allows that the concrete behaviors take more (or possibly less) steps than the abstract ones.

Eternity variables are introduced as a more powerful alternative for prophecy variables and backward simulations. This formalism is semantically complete: every simulation that preserves quiescence is a composition of a forward simulation, an extension with eternity variables, and a refinement mapping. This result does not need finite invisible nondeterminism and machine closure as in the Abadi–Lamport Theorem. The requirement of internal continuity is weakened to preservation of quiescence.

Almost all concepts are illustrated by tiny examples or counter-examples.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computing—*Automata (e.g., finite, push-down, resource-bounded)*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal logic*

General Terms: Theory, Verification

Additional Key Words and Phrases: History variables, implementation, invariant, preservation of quiescence, prophecy variables, refinement mapping, simulation

1. INTRODUCTION

We propose eternity variables as a new formal tool to verify concurrent and distributed algorithms. Similar variables may have been used informally in the past in verifications as for example, Broy [1992]. Eternity variables can also be applied to improve the abstractness and conciseness of specifications [Hesselink 2004]. It is likely that they can be transferred to input-output automata, labeled transition systems, and perhaps even real-time and hybrid systems.

Author's address: Dept. of Mathematics and Computing Science, Rijksuniversiteit Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands; email: wim@cs.rug.nl, Web: http://www.cs.rug.nl/~wim.

Apart from proposing eternity variables and proving their soundness and completeness, this article may serve as an introduction to the various forms of simulation for not necessarily terminating programs. We illustrate almost all concepts by tiny toy examples to sharpen the intuition.

1.1 Auxiliary Variables

Eternity variables form a new kind of auxiliary variables, variables that are added to a program to argue about it. Auxiliary variables occur when, in order to analyse a program, say K, one extends it with auxiliary variables and actions upon them to a bigger program, say L, proves some property of L, and infers something for the program, K, without them.

Since the Seventies, auxiliary variables have been used to prove the correctness of concurrent systems, for example, Clint [1973] and Owicki and Gries [1976]. These auxiliary variables served to record the history of the system's behavior. They are therefore sometimes called history variables. In for example, de Roever et al. [2001], it is proved that they are sufficient to prove that a terminating concurrent system satisfies a specification in terms of pre- and postconditions. Such a result is called semantic completeness.

In this article, we want to allow nonterminating programs and therefore use "abstract" programs as specifications. The correctness issue then becomes the question of the implementation relation between programs. Over the years, the idea of implementation has been formalized in many different settings, under names like refinement and simulation.

In or before 1986, it was proved that the combination of forward and backward simulations was sufficient to prove "data refinement" for terminating programs [He et al. 1986]. In 1988, Abadi and Lamport [1991] proposed prophecy variables to guess future behavior of nonterminating programs. They proved that the combination of history variables, prophecy variables and refinement mappings is—in a certain sense—sufficient to prove arbitrary implementation relations between nonterminating programs. Although refinement mappings and extension with history variables can be regarded as forward simulations, and prophecy variables correspond to backward simulations, the two proofs of semantic completeness are very different and the two papers [Abadi and Lamport 1991; He et al. 1986] do not refer to each other. They even have disjoint bibliographies.

The soundness of prophecy variables relies on König's Lemma; therefore, application of them requires that the invisible (i.e., internal) nondeterminism of the system is finite. One may argue that imposing finiteness should be acceptable since computer storage is always finite. Consider however the case that the prophecy would be the guess of a sequence number for the transactions in a reactive system, say an operating system or a database. Without a bound on the numbers, the choice would be infinite, but it is inacceptable to impose a bound on the number of transactions in the specification of such a system. Indeed, one would rather specify that the system can proceed indefinitely. Below, we give an example (3.8) to show the unsoundness of prophecy variables with a relation that allows infinite choices.

We therefore develop an alternative for prophecy variables that does not rely on König's Lemma. The eternity variable we propose as an alternative, is less flexible and it is chosen only once, nondeterministically and before the computation starts. Its value must of course be related to the behavior as it develops. This will be dealt with in the so-called behavior restriction. The proof of soundness for extension with eternity variables with a valid behavior restriction is much easier than for prophecy variables.

The new combination of extension with eternity variables and forward simulations is also proved to be semantically complete. This proof is somewhat easier than the corresponding proof for prophecy variables. We actually have two versions of this result, which differ in the degree of ignoring stutterings.

1.2 Additional Technical Assumptions

Our setting is the theory of Abadi and Lamport [1991], where programs, systems, and specifications are all regarded as specifications. A specification is a state machine with a supplementary property. Behaviors of a specification are infinite sequences of states. Behaviors become visible by means of an observation function. A specification implements another one when all visible behaviors of the first one can occur as visible behaviors of the second one. Although they can change roles, let us call the implementing specification the concrete one and the implemented specification the abstract one.

Under some technical assumptions, Abadi and Lamport [1991] proved that, when a specification K implements a specification L, there exists an extension M of K with history variables and prophecy variables together with a refinement mapping from M to L. The assumptions needed are that K should be "machine closed", and that L should be "internally continuous" and of "finite invisible nondeterminism".

In our alternative with eternity variables instead of prophecy variables, "internal continuity" is weakened to "preservation of quiescence" while the other two assumptions are eliminated. Preservation of quiescence means that, whenever the concrete specification can repeat the current state indefinitely, the abstract specification is allowed to do so as well. In other words, when the implementation stops, the specification allows this. Preservation of quiescence is quite common. Indeed, refinement mappings and extensions with history, prophecy or eternity variables all preserve quiescence.

1.3 Stuttering Behavior

Since the concrete specification may have to perform computation steps that are not needed for the abstract specification, we follow [Abadi and Lamport 1991; Lamport 1994] by allowing all specifications to stutter: a behavior remains a behavior when a state in it is duplicated.

In Abadi and Lamport [1991], it is also allowed that the concrete specification is faster than the abstract one: a concrete behavior may have to be slowed down by adding stutterings in order to match some abstract behavior. This may seem questionable since one may argue that, when the concrete specification needs fewer steps than the abstract one, the abstract one is not abstract enough. Yet,

experience shows that there need not be anything wrong with a specification when the implementation can do with fewer steps [Lamport 1989].

We therefore developed two theories: a strict theory and a stuttering theory Hesselink [2005]. The stuttering theory corresponds to the setting of Abadi and Lamport [1991], where the concrete specification can do both more and fewer steps than the abstract specification. In the strict theory, the concrete specification can do more but not fewer steps than the abstract specification. This results in a hierarchy of implementations that is finer than for the stuttering theory. In this article we only present the strict theory, since it is simpler and more elegant than the stuttering theory of Hesselink [2005].

1.4 Simulations of Specifications

A refinement mapping is a function between the states that, roughly speaking, preserves the initial states, the next-state relation and the supplementary property. Adding history or prophecy variables to the state gives rise to forward and backward simulations.

We unify these three concepts by introducing simulations. Actually, the term "simulation" has been introduced by Milner [1971]. He used it for a kind of relation, which was later called downward or forward simulation to distinguish it from so-called upward or backward simulation [He et al. 1986; Lynch and Vaandrager 1995]. It seems natural and justified to reintroduce the term "simulation" for the common generalization.

Our simulations are certain binary relations. For the sake of simplicity, we treat binary relations as sets of pairs, with some notational conventions. Since we use $X \to Y$ for functions from X to Y, and $P \Rightarrow Q$ for implication between predicates P and Q, we write $F : K \mathrel{\rightarrowtriangle} L$ to denote that relation F is a simulation of specifications from K to L. We hope the reader is not confused by the totally unrelated arrows $\mathrel{\rightarrowtriangle}$ used by Abadi and Lamport [1995].

The notation $F : K \mathrel{\rightarrowtriangle} L$ is inspired by category theory. Indeed, specifications with their simulations form the objects and morphisms of a category. Categories were introduced in mathematics by Eilenberg and MacLane [1945]. Since every introduction to category theory goes far beyond our needs, we refrain from further references.

Our first main result is a completeness theorem: A specification implements another one if and only if there is a certain simulation between them. This shows that our concept of simulation is general enough to capture the relevant phenomena.

1.5 Eternity Variables and Completeness

In the field of program verification, simulations serve to prove correctness, that is, the existence of an implementation relation between a program and a specification. The idea of refinement calculus is to construct simulations by composing them. Refinement mappings and forward simulations are the main candidates, but they are not enough. In general, one also needs simulations with kind of "prescient behavior" as exhibited by backward simulations. It is at this point that our eternity variables come in.

An eternity variable is a kind of logical variable with a value constrained by the current execution. Technically, it is an auxiliary variable, which may be initialized nondeterministically and is never modified thereafter. Its value is constrained by a relation with the state. A behavior that would violate such a constraint, is discarded. The verifier of a program has to prove that the totality of constraints is not contradictory. For example, the eternity variable can be an infinite array while the conditions constrain different elements of it.

The simulation from the original specification to the one obtained by extending it with the eternity variable is called the eternity extension. We thus have four basic kinds of simulations: refinement mappings, forward simulations, backward simulations, and eternity extensions. Every composition of simulations is a simulation. If relation G contains a simulation $K \rightarrow\!\!\triangleright L$, then G itself is a simulation $K \rightarrow\!\!\triangleright L$. Therefore, in order to prove that some relation G is a simulation $K \rightarrow\!\!\triangleright L$, it suffices to find basic simulations such that the composition of them is contained in G. The completeness result is that, conversely, every simulation that preserves quiescence contains a composition of a forward simulation, an eternity extension, and a refinement mapping.

More specifically, every specification K has a so-called unfolding $K^\#$ [Lynch and Vaandrager 1995] with a forward simulation $K \rightarrow\!\!\triangleright K^\#$. Given a simulation $F : K \rightarrow\!\!\triangleright L$ that preserves quiescence, we construct an intermediate specification W as an extension of $K^\#$ with an eternity variable, together with a refinement mapping $W \rightarrow\!\!\triangleright L$, such that the composition of the simulations $K \rightarrow\!\!\triangleright K^\#$ and $K^\# \rightarrow\!\!\triangleright W$ and $W \rightarrow\!\!\triangleright L$ is a subset of relation F.

When one wants to use eternity variables to prove some simulation relation, application of the unfolding $K^\#$ is overkill. Instead, one introduces approximating history variables to collect the relevant parts of the history. In Section 4.3, we briefly discuss the methodological issues involved. A complete, but still tiny example is treated in Section 5. We refer to Hesselink [2004] for an actual application.

1.6 Overview

In Section 1.7, we briefly discuss related work. Section 1.8 contains technical material on relations and lists. We treat stuttering and temporal operators in Section 1.9. In Section 2, we introduce specifications and simulations, and prove the characterizing theorem for them. In Section 3, we present the theory of forward and backward simulations in our setting and introduce quiescence and preservation of quiescence. Eternity variables are introduced in Section 4, where we also prove soundness and semantic completeness for eternity variables in the strict theory. Section 5 contains a tiny application of the method: we consider a relation between the state spaces of two specifications and prove that it is a simulation by factoring it over a forward simulation, an eternity extension, an invariant restriction and two refinement mappings. Conclusions are drawn in Section 6.

A preliminary version of the theory [Hesselink 2002b] was presented at MPC 2002. This article is flawed by an incorrect completeness theorem; we only saw

the need of preservation of quiescence some weeks before the conference when the proceedings were already in print.

New concepts in this article are simulation, preservation of quiescence, and eternity extension. New results are the completeness theorem of simulation with respect to implementation in Section 2.4, the relationship between internal continuity and preservation of quiescence in Section 3.5, and the soundness and completeness of eternity extensions in Section 4.

1.7 Related Work

Our primary inspiration was Abadi and Lamport [1991]. Our formalism is a semantical version of Lamport's TLA [Lamport 1994]. Lynch and Vaandrager [1995] and Jonnson [1991] present forward and backward simulations and the associated results on semantic completeness in the closely related settings of untimed automata and fair labeled transition systems. Our investigation was triggered by Cohen and Lamport [1998] on Lipton's Theorem [Lipton 1975] about refining atomicity. While working on the serializable database interface problem of Lamport [1992] and Schneider [1992], we felt the need for variables with "prescient" behavior without finiteness assumptions. This led us to the invention of eternity variables, which we applied successfully in the mean time to the serializable database interface in Hesselink [2004]. Jonsson et al. [1999] present another way of proving refinement that avoids the finiteness assumptions of backward simulations. They use a very flexible concept of refinement based on so-called pomsets, but have no claim of semantic completeness.

1.8 Relations and Lists

We treat a binary relation as a set of pairs. So, a binary relation between sets X and Y is a subset of the Cartesian product $X \times Y$. We use the functions *fst* and *snd* given by $fst(x, y) = x$ and $snd(x, y) = y$. A binary relation on X is a subset of $X \times X$. The identity relation 1_X on X consists of all pairs (x, x) with $x \in X$. Recall that a binary relation A on X is called *reflexive* iff $1_X \subseteq A$. The *converse* $cv(A)$ of a binary relation A is defined by $cv(A) = \{(x, y) \mid (y, x) \in A\}$.

For binary relations A and B, the composition $(A; B)$ is defined to consist of all pairs (x, z) such that there exists y with $(x, y) \in A$ and $(y, z) \in B$. A function $f : X \to Y$ is identified with its graph $\{(x, f(x)) \mid x \in X\}$ which is a binary relation between X and Y. The composition of functions $f : X \to Y$ and $g : Y \to Z$ is a function $g \circ f : X \to Z$, which equals the relational composition $(f; g)$.

We use lists to represent consecutive values during computations. If X is a set, we write X^+ for the set of the nonempty finite lists and X^ω for the set of infinite lists over X. We write $\ell(xs)$ for the length of list xs. The elements of xs are xs_i for $0 \leq i < \ell(xs)$. If xs is a list of length $\ell(xs) \geq n$, we define $(xs \mid n)$ to be its prefix of length n. We write $xs \sqsubseteq xt$ to denote that list xs is a prefix of xt, possibly equal to xt. We define $last : X^+ \to X$ to be the function that returns the last element of a nonempty finite list.

A function $f : X \to Y$ induces a function $f^\omega : X^\omega \to Y^\omega$. For a binary relation $F \subseteq X \times Y$, we have an induced binary relation $F^\omega \subseteq X^\omega \times Y^\omega$ given by

$$(xs, ys) \in F^\omega \quad \equiv \quad (\forall\, i :: (xs_i, ys_i) \in F)\ .$$

1.9 Stuttering and Properties

Let P be a set of infinite lists over X, that is, a subset of X^ω. We write $\neg P$ to denote the complement (negation) of P. For an infinite list xs, we write $Suf(xs)$ to denote the set of its infinite suffixes. The sets $\Box P$ (always P), and $\Diamond P$ (sometime P) are defined by

$$\begin{array}{lcl} xs \in \Box P & \equiv & Suf(xs) \subseteq P\ , \\ \Diamond P & = & \neg\Box\neg P\ . \end{array}$$

So, $xs \in \Box P$ means that all suffixes of xs belong to P, and $xs \in \Diamond P$ means that xs has some suffix that belongs to P.

For $U \subseteq X$ and $A \subseteq X \times X$, the subsets $[\![U]\!]$ and $[\![A]\!]$ of X^ω are defined by

$$\begin{array}{lcl} xs \in [\![U]\!] & \equiv & xs_0 \in U\ , \\ xs \in [\![A]\!] & \equiv & (xs_0, xs_1) \in A\ . \end{array}$$

So, $[\![U]\!]$ consists of the infinite lists that start in U, and $[\![A]\!]$ consists of the infinite lists that start with an A-transition.

We define a list xs to be an *unstuttering* of a list ys, notation $xs \preceq ys$, iff xs is obtained from ys by replacing some finite nonempty subsequences ss of consecutive equal elements of ys with their first elements ss_0. The number of such subsequences that are replaced may be infinite. For example, if, for a finite list vs, we write vs^ω to denote the list obtained by concatenating infinitely many copies of vs, the list $(abbccb)^\omega$ is an unstuttering of $(aaabbbccb)^\omega$.

A finite list xs is called *stutterfree* iff every pair of consecutive elements differ. An infinite list xs is called *stutterfree* iff it stutters only after reaching a final state, i.e., iff $xs_i = xs_{i+1}$ implies $xs_{i+1} = xs_{i+2}$ for all i. For every infinite list xs, there is a unique stutterfree infinite list xt with $xt \preceq xs$. For example, if $xs = (aaabbbccb)^\omega$ then $xt = (abcb)^\omega$.

A subset P of X^ω is called a *property over* X iff $xs \preceq ys$ implies that $xs \in P \equiv ys \in P$. This definition is equivalent to the one of Abadi and Lamport [1991]. If P is a property, then $\neg P$, $\Box P$, and $\Diamond P$ are properties. If U is a subset of X, then $[\![U]\!]$ is a property. If A is a reflexive relation on X, then $\Box[\![A]\!]$ is a property, and it consists of the infinite lists with all transitions belonging to A.

2. SPECIFICATIONS AND SIMULATIONS

In this section we introduce the central concepts of the theory. Following Abadi and Lamport [1991], we define specifications in Section 2.1. Refinement mappings are introduced in Section 2.2. In 2.3, we define simulations. In Section 2.4, we define visible specifications and their implementation relations, and we prove that simulations characterize the implementations between visible specifications.

2.1 Specifications

A *specification* is defined to be a tuple $K = (X, Y, N, P)$ where X is a set, Y is a subset of X, N a reflexive binary relation on X, and P is a property over X. The set X is called the *state space*, its elements are called *states*, the elements of Y are called *initial states*. Relation N is called the *next-state* relation. The set P is called the *supplementary* property.

We define an *initial execution* of K to be a nonempty list xs over X with $xs_0 \in Y$ and such that every pair of consecutive elements belongs to N. We define a *behavior* of K to be an infinite initial execution xs of K with $xs \in P$. We write $Beh(K)$ to denote the set of behaviors of K.

The triple (X, Y, N) can be regarded as a state machine [Abadi and Lamport 1991]. The supplementary property P is often used for fairness conditions but can also be applied for other purposes. The initial executions of K are determined by the state machine. The supplementary property is a restriction on the behaviors.

It is easy to see that $Beh(K) = [\![Y]\!] \cap \Box[\![N]\!] \cap P$. It follows that $Beh(K)$ is a property. The requirement that relation N is reflexive is imposed to allow stuttering: if xs is a behavior of K, any list ys obtained from xs by repeating elements of xs or by removing subsequent duplicates is also a behavior of K. In particular, for every behavior xs of K, there is a unique stutterfree behavior xt of K with $xt \preceq xs$.

The components of specification $K = (X, Y, N, P)$ are denoted $states(K) = X$, $start(K) = Y$, $step(K) = N$ and $prop(K) = P$.

Specification K is defined to be *machine closed* [Abadi and Lamport 1991] iff every finite initial execution of K can be extended to a behavior of K. We would encourage specifiers to write specifications that are not machine closed whenever that improves clarity, for example, see Ladkin et al. [1999, Section 3.2.3]. If the specification is not machine closed, it is important to distinguish between states reachable from initial states and states that occur in behaviors.

We therefore define a state of K to be *reachable* iff it occurs in an initial execution of K, and to be *occurring* iff it occurs in a behavior of K. A subset of $states(K)$ is called a *forward invariant* iff it contains all reachable states. It is called an *invariant* iff it contains all occurring states. Recall that a subset is called a *strong invariant* (or *inductive* [Manna and Pnueli 1995]) iff it contains all initial states and is preserved in every step, that is, J is a strong invariant iff $Y \subseteq J$ and $y \in J$ for every pair $(x, y) \in N$ with $x \in J$. It is easy to see that every strong invariant is a forward invariant and that every forward invariant is an invariant.

Example 2.1. Reachable states need not be occurring, an invariant need not be a forward invariant, and a forward invariant need not be a strong invariant. This is shown by the following program:

var $\mathtt{k} : Int := 0$;
do $\mathtt{k} = 0 \;\rightarrow\; choose\ \mathtt{k} > 0$;
[] $\mathtt{k} \neq 0 \;\rightarrow\; \mathtt{k} := \mathtt{k} - 2$;
od ;
prop: infinitely often $\mathtt{k} = 0$.

Note that this program only stands for a specification. It is not supposed to be directly executable.

Formally, the specification is (X, Y, N, P) where X is the set of the integers and $Y = \{0\}$. A pair (k, k') belongs to relation $N \subseteq X \times X$ if and only if

$$(k = 0 \ \wedge \ k' > 0) \quad \vee \quad (k \neq 0 \ \wedge \ k' = k - 2) \quad \vee \quad k' = k.$$

The third disjunct serves to allow stuttering. Property P consists of the infinite sequences with infinitely many zeroes, that is, $P = \Box\Diamond[\![Y]\!]$. It follows that the only occurring states are the even natural numbers. So, the even natural numbers form an invariant $J0$. The set of the natural numbers is also an invariant. The set of reachable states is $J1 = \{k \mid k \geq 0 \ \vee \ k \ \mathbf{mod}\ 2 = 1\}$. Therefore, $J0$ is not a forward invariant. The set $J1 \cup \{-2\}$ is a forward invariant but not a strong invariant, since there is a step from -2 to -4. □

2.2 Refinement Mappings

Let K and L be specifications. A function $f : \mathit{states}(K) \to \mathit{states}(L)$ is called a *refinement mapping* [Abadi and Lamport 1991] from K to L iff $f(x) \in \mathit{start}(L)$ for every $x \in \mathit{start}(K)$, and $(f(x), f(x')) \in \mathit{step}(L)$ for every pair $(x, x') \in \mathit{step}(K)$, and $f^{\omega}(xs) \in \mathit{prop}(L)$ for every $xs \in \mathit{Beh}(K)$. Refinement mappings form the simplest way to compare different specifications.

Example 2.2. For $m > 1$, let $K(m)$ be the specification that corresponds to the program

var j : *Nat* := 0 ;
do *true* → j := (j + 1) **mod** *m* **od** ;
prop: j changes infinitely often.

We thus have $\mathit{states}(K(m)) = \mathbb{N}$, $\mathit{start}(K(m)) = \{0\}$, $\mathit{prop}(K(m)) = \Box\Diamond[\![\neq]\!]$, and

$$(j, j') \in \mathit{step}(K(m)) \quad \equiv \quad j' \in \{j, (j + 1) \ \mathbf{mod}\ m\}.$$

In order to give an example of a refinement mapping, we regard $K(20)$ as an implementation of $K(13)$. Let $f : \mathbb{N} \to \mathbb{N}$ be the function given by $f(j) = \min(j, 12)$. It is easy to verify that f is a refinement mapping from $K(20)$ to $K(13)$. Note that the abstract behavior (in $K(13)$) stutters whenever the concrete behavior (in $K(20)$) is proceeding from 12 to 19. This example shows that it is useful that the next-state relation is always reflexive. □

2.3 Simulations

Recall from 1.8 that a relation F between $\mathit{states}(K)$ and $\mathit{states}(L)$ induces a relation F^{ω} between the sets of infinite lists $(\mathit{states}(K))^{\omega}$ and $(\mathit{states}(L))^{\omega}$.

We define relation F to be a *simulation* $K \rightarrowtriangle L$ iff, for every behavior $xs \in \mathit{Beh}(K)$, there exists a behavior $ys \in \mathit{Beh}(L)$ with $(xs, ys) \in F^{\omega}$. The following two examples show that refinement mappings are not enough and that simulations are useful.

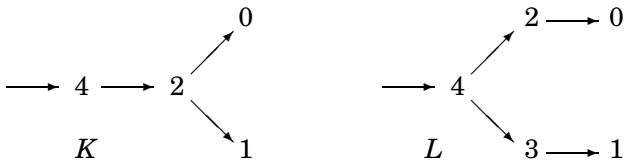
Example 2.3. We use the specifications $K(m)$ and $K(2 \cdot m)$ in accordance with Example 2.2. Let the binary relation F be given by

$$(j,k) \in F \quad \equiv \quad j = k \ \mathbf{mod}\ m \ .$$

Then F is a simulation $K(m) \rightarrowtriangle K(2 \cdot m)$, but there is no refinement mapping from $K(m)$ to $K(2 \cdot m)$. □

Example 2.4. We consider two specifications K and L, both with state space $X = \{0, 1, 2, 3, 4\}$, initial set $Y = \{4\}$, and property $\diamond : [\![\{0, 1\}]\!]$. The next-state relations are

$$\begin{array}{lcl} \mathit{step}(K) & = & 1_X \cup \{(4,2),(2,1),(2,0)\} \ , \\ \mathit{step}(L) & = & 1_X \cup \{(4,3),(4,2),(3,1),(2,0)\} \ . \end{array}$$



Both specifications have the final outcomes 0 and 1, but K postpones the choice, while L chooses immediately. We regard only the final states 0 and 1 as visible. The stutterfree behaviors of K are $(4, 2, 0^\omega)$ and $(4, 2, 1^\omega)$, while those of L are $(4, 2, 0^\omega)$ and $(4, 3, 1^\omega)$. Therefore, K and L implement each other. One can easily verify that relation $F = 1_X \cup \{(2, 3)\}$ is a simulation $F : K \rightarrowtriangle L$. There is no refinement mapping f from K to L with $f(0) = 0$ and $f(1) = 1$, since the concrete specification K makes the choice between the outcomes later than the abstract specification L. At concrete state 2, simulation F "needs prescience" to choose between the abstract states 2 and 3. □

In general, it should be noted that the mere existence of a simulation $F : K \rightarrowtriangle L$ does not imply much. If $F : K \rightarrowtriangle L$ and G is a relation with $F \subseteq G$, then $G : K \rightarrowtriangle L$. Therefore, the smaller the simulation, the more information it carries. It is easy to verify that simulations can be composed: if F is a simulation $K \rightarrowtriangle L$ and G is a simulation $L \rightarrowtriangle M$, the composed relation $(F;G)$ is a simulation $K \rightarrowtriangle M$. It is also easy to verify that a refinement mapping $f : \mathit{states}(K) \rightarrow \mathit{states}(L)$, when regarded as a relation as in Section 1.8, is a simulation $K \rightarrowtriangle L$.

We often encounter the following situation. A specification L is regarded as an extension of specification K with a variable of a type M iff $\mathit{states}(L)$ is (a subset of) the Cartesian product $\mathit{states}(K) \times M$ and the function $\mathit{fst} : \mathit{states}(L) \rightarrow \mathit{states}(K)$ is a refinement mapping. The second component of the states of L is then regarded as the variable added. The extension is called a refinement extension iff the converse $\mathit{cv}(\mathit{fst})$ is a simulation $K \rightarrowtriangle L$.

2.4 Visibility and Completeness of Simulation

We are usually not interested in all details of the states, but only in certain aspects of them. This means that there is a function from $\mathit{states}(K)$ to some other set that we regard as an observation function. A *visible specification* is

therefore defined to be a pair (K, f) where K is a specification and f is some function defined on $states(K)$. Deviating from Abadi and Lamport [1991], we define the set of observations by

$$Obs(K, f) \quad = \quad \{f^{\omega}(xs) \mid xs \in Beh(K)\} .$$

Note that $Obs(K, f)$ need not be a property. If xs is an observation and $ys \preceq xs$, then ys need not be an observation.

Example 2.5. Assume we are observing $K(13)$ of Example 2.2 with the test $j > 0$. So, we use the observation function $f(j) = (j > 0)$. Then the observations are the Boolean lists with infinitely many values *true* and infinitely many values *false*, in which every *true* stutters at least 12 times. □

Let (K, f) and (L, g) be visible specifications with the functions f and g mapping to the same set. Then (K, f) is said to *implement* (L, g) iff $Obs(K, f)$ is contained in $Obs(L, g)$, that is, iff for every $xs \in Beh(K)$ there exists $ys \in Beh(L)$ with $f^{\omega}(xs) = g^{\omega}(ys)$. This concept of implementation is stronger than that of Abadi and Lamport [1991]: we do not allow that an observation of (K, f) can only be mimicked by (L, g) after inserting additional stutterings.

Our concept of simulation is motivated by the following completeness theorem, the proof of which is rather straightforward.

THEOREM 2.6. *Consider visible specifications* (K, f) *and* (L, g) *where* f *and* g *are functions to the same set. We have that* (K, f) *implements* (L, g) *if and only if there is a simulation* $F : K \rightarrowtriangle L$ *with* $(F; g) \subseteq f$.

PROOF. The proof is by mutual implication.

First, assume the existence of a simulation $F : K \rightarrowtriangle L$ with $(F; g) \subseteq f$. Let $zs \in Obs(K, f)$. We have to prove that $zs \in Obs(L, g)$. By the definition of Obs, there exists $xs \in Beh(K)$ with $zs = f^{\omega}(xs)$. Since F is a simulation, there exists $ys \in Beh(L)$ with $(xs, ys) \in F^{\omega}$. For every number n, we have $(xs_n, ys_n) \in F$ and, hence, $(xs_n, g(ys_n)) \in (F; g) \subseteq f$ and, hence, $g(ys_n) = f(xs_n) = zs_n$. This implies that $zs = g^{\omega}(ys) \in Obs(L, g)$.

Next, assume that (K, f) implements (L, g). We define relation F between $states(K)$ and $states(L)$ by $F = \{(x, y) \mid f(x) = g(y)\}$. For every pair $(x, z) \in (F; g)$ there exists y with $(x, y) \in F$ and $(y, z) \in g$; we then have $f(x) = g(y) = z$. This proves $(F; g) \subseteq f$. It remains to prove that F is a simulation $K \rightarrowtriangle L$. Let $xs \in Beh(K)$. Since $Obs(K, f) \subseteq Obs(L, g)$, there is $ys \in Beh(L)$ with $f^{\omega}(xs) = g^{\omega}(ys)$. We thus have $(xs, ys) \in F^{\omega}$. This proves that F is a simulation $K \rightarrowtriangle L$. □

Example 2.7. Consider the visible specifications (K, f) and (L, g) with $K = K(m)$ and $L = K(2 \cdot m)$ as in Example 2.3, with $f, g : \mathbb{N} \rightarrow \mathbb{N}$ given by $f(j) = j$ and $g(j) = j$ **mod** m. Then relation F as constructed in the above proof equals relation F of Example 2.3. □

3. SPECIAL SIMULATIONS

In this section, we introduce forward and backward simulations as special kinds of simulations. Forward simulations are introduced in 3.1. They correspond to

refinement mappings and to the well-known addition of history variables. In 3.2, we show that invariants give rise to simulations. In Section 3.3, we introduce the unfolding [Lynch and Vaandrager 1995] of a specification, which plays a key role in several proofs of semantic completeness. Backward simulations are introduced in Section 3.4. Quiescence and preservation of quiescence are introduced in Section 3.5.

3.1 Flatness and Forward Simulations

We start with a technical definition concerning the supplementary property of the related specifications. A relation F between $states(K)$ and $states(L)$ is defined to be *flat from K to L* iff every infinite initial execution ys of L with $(xs, ys) \in F^\omega$ for some $xs \in Beh(K)$ satisfies $ys \in prop(L)$.

It turns out that all our basic kinds of simulations are flat. Indeed, refinement mappings are flat and we need flatness as a defining condition for both forward and backward simulations. Flatness always serves as the finishing touch in the construction of the abstract behavior. Yet, flatness is not a nice property: in Example 3.3 below, we show that the composition of two flat simulations need not be flat.

The easiest way to prove that one specification simulates (the behavior of) another is by starting at the beginning and constructing the corresponding behavior in the other specification inductively. This requires a condition embodied in so-called forward or downward simulations [He et al. 1986; Lynch and Vaandrager 1995], which go back at least to Milner [1971]. They are defined as follows:

A relation F between $states(K)$ and $states(L)$ is defined to be a *forward simulation* from specification K to specification L iff

(F0) For every $x \in start(K)$, there is $y \in start(L)$ with $(x, y) \in F$.

(F1) For every pair $(x, y) \in F$ and every x' with $(x, x') \in step(K)$, there is y' with $(y, y') \in step(L)$ and $(x', y') \in F$.

(F2) Relation F is flat from K to L.

Example 3.1. It is easy to verify that relation F of Example 2.3 is a forward simulation. Every refinement mapping, when regarded as a relation, is also a forward simulation. □

The definition of forward simulations is justified by the following well-known result:

LEMMA 3.2. *Every forward simulation F from K to L is a simulation $K \rightarrowtriangle L$.*

PROOF. Let $xs \in Beh(K)$ be given. Then $xs_0 \in start(K)$, so by (F0), there is $ys_0 \in start(L)$ with $(xs_0, ys_0) \in F$. Since $(xs_n, xs_{n+1}) \in step(K)$ for all n, we can use (F1) inductively to construct an infinite initial execution ys of L that satisfies $(xs_n, ys_n) \in F$ for all n. Since relation F is flat, we conclude that ys is a behavior of L with $(xs, ys) \in F^\omega$. Therefore, F is a simulation $K \rightarrowtriangle L$. □

Example 3.3. Let $X = [0 \cdots N]$ for some number $N \geq 2$. Let K be the specification with the program

var k : $X := 0$;
do *true* → *choose* k ∈ X **od**/ ;
prop: k changes infinitely often and is sometimes 1.

So, we have $states(K) = X$, $start(K) = \{0\}$, and $step(K) = X^2$. The property $prop(K)$ is the intersection of $\Box\Diamond[\![\neq]\!]$ and $\Diamond[\![\mathtt{k} = 1]\!]$.

Let L be the specification with

var j : $X := 0$,
b : *Boolean* := *false* ;
do *true* → *choose* j ∈ X ;
[] j = 1 → b := *true* ; *choose* j ∈ X;
od ;
prop: b is sometimes *true*.

In such programs, we regard the alternatives in the **do** loop as atomic. So we have

$$((j, b), (j', b')) \in step(L) \quad \equiv \quad (b' = b) \ \vee \ (j = 1 \ \wedge \ b') \, .$$

The property is $prop(L) = \Diamond[\![\mathtt{b}]\!]$.

It is easy to show that relation $F = \{(k, (j, b)) \mid k = j\}$ is a simulation $K \rightarrowtriangle L$. Indeed, let xs be a behavior of K. Then there is an index r with $xs_r = 1$. Let ys be the sequence in $states(L)$ given by $ys_i = (xs_i, (r < i))$ for all i. Since the boolean component b of ys becomes true in a step with precondition $\mathtt{j} = 1$, this is a behavior of L, which satisfies $(xs, ys) \in F^\omega$. Simulation $F : K \rightarrowtriangle L$ is not flat, since the sequence zs with $zs_i = (xs_i, false)$ for all i is not a behavior of L but is an infinite initial execution of L with $(xs, zs) \in F^\omega$.

In order to show that F is a composition of two forward simulations, we make specification L more deterministic. Let L' be the specification obtained from L by restricting the step relation to

do j ≠ 1 → *choose* j ∈ X ;
[] j = 1 → b := *true* ; *choose* j ∈ X ;
od .

Since stuttering must be allowed, a pair $((j, b), (j', b'))$ belongs to $step(L')$ if and only if

$$b' = (b \ \vee \ j = 1) \quad \vee \quad (j = j' \ \wedge \ b = b') \, .$$

The above relation F is a forward simulation $K \rightarrowtriangle L'$. Indeed, condition (F0) is obvious. Condition (F1) holds since every step of K can be mimicked by L'. Flatness is shown as follows. Let xs be a behavior of K. The property of K implies that there is an index r with $xs_r = 1 \neq xs_{r+1}$. If ys is an infinite initial execution of L' with $(xs, ys) \in F^\omega$, then ys is a behavior of L' since $ys_{r+1} = (xs_{r+1}, true)$.

It is easy to verify that the identity function id is a refinement mapping $L' \rightarrowtriangle L$ and hence a forward simulation. The simulation $F : K \rightarrowtriangle L$ is clearly the composition $F = (F; id)$. So, here we have indeed a nonflat composition of two forward simulations. □

3.2 Invariant Restriction

Invariants are often used to restrict the state space implicitly. When the state space is made explicit, restriction to an invariant subspace turns out to be a simulation.

Slightly more general, let D be a subset of $states(K)$ for a specification K. Then, we can define the D-restricted specification K_D by $states(K_D) = D$ and $start(K_D) = D \cap start(K)$ and $step(K_D) = D^2 \cap step(K)$ and $prop(K_D) = D^\omega \cap prop(K)$. Indeed, it is easy to verify that $step(K_D)$ is reflexive and that $prop(K_D)$ is a property. The following result characterizes invariants via simulations.

LEMMA 3.4

(a) *The identity relation* 1_D *is a simulation* $K \rightarrowtriangle K_D$ *if and only if* D *is an invariant.*

(b) 1_D *is a forward simulation* $K \rightarrowtriangle K_D$ *if and only if* D *is a strong invariant.*

We skip the proof, since it is fairly straightforward and not interesting.

3.3 The Unfolding

The *unfolding* $K^\#$ of a specification K plays a key role in the proofs of semantic completeness in Abadi and Lamport [1991] and Lynch and Vaandrager [1995] as well as in our semantic completeness result below.

It is defined as follows: $states(K^\#)$ consists of the stutterfree finite initial executions of K. The initial set $start(K^\#)$ consists of the elements $xs \in states(K^\#)$ with $\ell(xs) = 1$. The next-state relation $step(K^\#)$ and the property $prop(K^\#) \subseteq (states(K^\#))^\omega$ are defined by

$$(xs, xt) \in step(K^\#) \quad \equiv \quad xs \sqsubseteq xt \ \wedge \ \ell(xt) \leq \ell(xs) + 1 \ ,$$
$$vss \in prop(K^\#) \quad \equiv \quad last^\omega(vss) \in prop(K) \ .$$

So, the nonstuttering steps of $K^\#$ are the pairs (xs, xt) with $xs \sqsubseteq xt$ and $\ell(xt) = \ell(xs) + 1$.

It is easy to prove that $K^\#$ is a specification. The function $last : states(K^\#) \rightarrow states(K)$ is a refinement mapping. Moreover, if $(xs, xt) \in step(K^\#)$ and $xs \neq xt$, then $last(xs) \neq last(xt)$ since xt is stutterfree. We are more interested, however, in the other direction. The following result of Abadi and Lamport [1991] is not difficult to prove.

LEMMA 3.5. *Relation* $cvl = cv(last)$ *is a forward simulation* $K \rightarrowtriangle K^\#$.

In Section 4.2 below, we shall need the following result.

LEMMA 3.6. *Let* $xs = last^\omega(vss)$ *for a stutterfree behavior* vss *of* $K^\#$. *Then* xs *is a behavior of* K *with* $vss_i \sqsubseteq xs$ *for all indices* i.

PROOF. Since vss is a behavior of $K^\#$, it is easy to verify that xs is a behavior of K. We now distinguish two cases. First, assume that $vss_i \neq vss_{i+1}$ for all i. Then $\ell(vss_i) = i + 1$ for all i. It follows that $vss_i = (xs \mid i + 1)$ for all i. Otherwise, let r be minimal with $vss_r = vss_{r+1}$. Since vss is stutterfree, $vss_i = vss_r$ for

all $i \geq r$. This implies $\ell(vss_i) = \min(i, r) + 1$ for all indices i. It follows that $vss_i = (xs \mid i + 1)$ for all i with $0 \leq i \leq r$ and $vss_i = (xs \mid r + 1)$ for all i with $r \leq i < \infty$. In either case, we have $vss_i \sqsubseteq xs$ for all indices i. □

3.4 Backward Simulations

It is also possible to prove that one specification simulates (the behavior of) another by starting arbitrarily far in the future and constructing a corresponding initial execution by working backwards. An infinite behavior is then obtained by a variation of König's Lemma. These so-called backward simulations [Lynch and Vaandrager 1995] form a relational version of the prophecy variables of Abadi and Lamport [1991] and are related to the upward simulations of He et al. [1986]. We give a variation of the version of Jonnson [1991].

Relation F between $states(K)$ and $states(L)$ is defined to be a *backward simulation* from K to L iff

(B0) Every pair $(x, y) \in F$ with $x \in start(K)$ satisfies $y \in start(L)$.

(B1) For every pair $(x', y') \in F$ and every x with $(x, x') \in step(K)$, there is y with $(x, y) \in F$ and $(y, y') \in step(L)$.

(B2) For every behavior xs of K there are infinitely many indices n for which the set $\{y \mid (xs_n, y) \in F\}$ is nonempty and finite.

(B3) Relation F is flat from K to L.

The simulation F presented in the Example 2.4 is a very simple example of a backward simulation. The verification of this is straightforward, though somewhat cumbersome.

An auxiliary variable added to the state space via a backward simulation is called a prophecy variable [Abadi and Lamport 1991] since it seems to show "prescient" behavior. In such a case, the relation is called a prophecy relation in Lynch and Vaandrager [1995]. The term *backward simulations* is justified by the following soundness result, the proof of which is a direct adaptation of the proof in Jonnson [1991].

LEMMA 3.7. *Every backward simulation F from K to L is a simulation $K \rightarrowtriangle L$.*

The empty relation $F = \emptyset$ always satisfies (B0), (B1), and (B3), but if K has any behavior, the empty relation is not a simulation from K to L. This justifies the nonemptyness condition in (B2). The following example shows that some finiteness in (B2) is also needed.

Example 3.8. The unsound doomsday prophet. Let L be the following extension of specification $K(13)$ of Example 2.2 with a natural variable k.

var j : *Nat* := 0 , k : *Nat* {*arbitrary*} ;
do k > 0 → j := (j + 1) **mod** 13 ; k := k − 1 **od** ;
prop: j changes infinitely often.

Since k cannot decrease infinitely often, j cannot change infinitely often. Therefore, specification L has no behaviors. Since $K(13)$ has behaviors, there cannot exist any simulation $K(13) \rightarrowtriangle L$. Function $fst : states(L) \rightarrow states(K(13))$ is a refinement mapping. Its converse, $F = cv(fst)$ cannot be a simulation $K(13) \rightarrowtriangle L$, but it is easily seen to satisfy (B0), (B1), and (B3). Indeed, it does not satisfy (B2) since $\{y \mid (x, y) \in F\}$ is infinite for every $x \in states(K(13))$.

The initial value of k can be regarded as a prophecy of doomsday, whence the name of the example. Note that $K(13)$ is deterministic and that the only nondeterminism in L is the infinite choice in the initialization. Also, note that we can restore condition (B2) by introducing a bound, say $k < 1000$, for the initial choice of k, but then condition (B1) is invalidated. □

3.5 Preservation of Quiescence

The completeness result of the next section needs the concept of "preservation of quiescence". Roughly speaking, a behavior is quiescent at a given state if it remains a behavior when the behavior after the state is replaced by an infinite repetition of the state. Preservation of quiescence means that the abstract behavior can be quiescent whenever the concrete behavior is quiescent. It is formalized as follows.

Given a natural number n and an infinite list xs, we define the infinite list $E_n(xs)$ as the concatenation of $(xs|n)$ with the infinite repetition of the state xs_n. We thus have $(E_n(xs))_k = xs_m$ where m is the minimum of n and k. A number n is a quiescent index of xs for specification K iff $E_n(xs)$ is a behavior of K. The set of quiescent indices of xs for K is defined as

$$Q_K(xs) = \{n \mid E_n(xs) \in Beh(K)\} .$$

Let K and L be specifications. A simulation $F : K \rightarrowtriangle L$ is said to *preserve quiescence* iff, for every $xs \in Beh(K)$, there exists $ys \in Beh(L)$ with $(xs, ys) \in F^\omega$ and $Q_K(xs) \subseteq Q_L(ys)$.

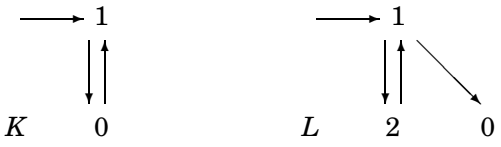
It is easy to verify that preservation of quiescence is compositional: if $F : K \rightarrowtriangle L$ and $G : L \rightarrowtriangle M$ both preserve quiescence, the composition $(F;G) : K \rightarrowtriangle M$ also preserves quiescence. Also, if $F : K \rightarrowtriangle L$ preserves quiescence and G is a relation between $states(K)$ and $states(L)$ with $F \subseteq G$, then G is a simulation $K \rightarrowtriangle L$ that preserves quiescence.

Example 3.9. Going back to Example 3.3, we let K' be the specification obtained from K by omitting the requirement that k keeps changing. So, the property is weakened to $prop(K') = \Diamond[\![\mathtt{k} = 1]\!]$. By the same argument as before, relation F is a simulation $K' \rightarrowtriangle L'$. This simulation does not preserve quiescence. Indeed, let xs and ys be behaviors of K' and L' with $(xs, ys) \in F^\omega$. Let r is the first index with $xs_r = 1$, then r is a quiescent index of xs but not of ys, since the Boolean b is still false. □

Example 3.10. We construct an even simpler simulation that does not preserve quiescence. Consider specifications K and L, both with state space

$X = \{0, 1, 2\}$, initial set $\{1\}$, and supplementary property $\Diamond\Box: [\![\{0\}]\!]$. The next-state relations are given by

$$\begin{aligned} step(K) &= 1_X \cup \{(1,0),(0,1)\}\ , \\ step(L) &= 1_X \cup \{(1,0),(1,2),(2,1)\}\ . \end{aligned}$$



The behaviors of K are infinite lists over $\{0, 1\}$ that start with 1 and contain only finitely many ones. The behaviors of L are finite lists over $\{1, 2\}$ that start and end with 1, followed by infinitely many zeroes. In either case, the quiescent indices are those of the zero elements in the list.

Let relation F on X be the set $F = \{(0,0),(0,2),(1,1)\}$. Relation F is a simulation $K \rightarrowtriangle L$. In fact, for every $xs \in Beh(K)$, there is precisely one $ys \in Beh(L)$ with $(xs, ys) \in F^\omega$. If n is the least number with $xs_i = 0$ for all $i \geq n$, then $ys_j = 2$ for all $j < n$ with $xs_j = 0$, and $ys_j = xs_j$ in all other cases. Since xs_j can be zero when ys_j is not, simulation F does not preserve quiescence. For instance, if $xs = (1, 0, 0, 1, 0^\omega)$, we need $ys = (1, 2, 2, 1, 0^\omega)$. □

Preservation of quiescence does not occur in Abadi and Lamport [1991]. Its role is played by the stronger concept of internal continuity. We therefore have to clarify the relationship between these concepts. Following Abadi and Lamport [1991], we define a visible specification (K, f) to be *internally continuous* iff every infinite initial execution xs of K with $f^\omega(xs) \in Obs(K, f)$ is a behavior of K. As the next result shows, internal continuity of the target specification implies preservation of quiescence by every simulation that yields an implementation according to Theorem 2.6.

THEOREM 3.11. *Let (K, f) and (L, g) be visible specifications and assume that (L, g) is internally continuous. Let $F : K \rightarrowtriangle L$ be a simulation with $(F; g) \subseteq f$. Then F preserves quiescence.*

PROOF. Let xs be a behavior of K. We have to provide a behavior ys of L with $(xs, ys) \in F^\omega$ and $Q_K(xs) \subseteq Q_L(ys)$. Since F is a simulation, we can choose a behavior ys of L with $(xs, ys) \in F^\omega$. It remains to prove that $Q_K(xs) \subseteq Q_L(ys)$.

Let $n \in Q_K(xs)$ be given. Write $xn = E_n(xs)$ and $yn = E_n(ys)$. Then yn is an infinite initial execution of L with $(xn, yn) \in F^\omega$. Just as in the proof of Theorem 2.6, the inclusion $(F; g) \subseteq f$ implies that $f(x) = g(y)$ for every $(x, y) \in F$. It follows that $g^\omega(yn) = f^\omega(xn)$. Since $n \in Q_K(xs)$, we have $f^\omega(xn) \in Obs(K, f)$. Theorem 2.6 implies that (K, f) implements (L, g). It therefore follows that $g^\omega(yn) \in Obs(L, g)$. Now, internal continuity of (L, g) implies that yn is a behavior of L, so that $n \in Q_L(ys)$. □

The following lemma implies that refinement mappings and forward and backward simulations all preserve quiescence.

LEMMA 3.12. *Every flat simulation $F : K \rightarrowtriangle L$ preserves quiescence.*

PROOF. Let $xs \in Beh(K)$. Since F is a simulation, there exists $ys \in Beh(L)$ with $(xs, ys) \in F^\omega$. It suffices to prove that $Q_K(xs) \subseteq Q_L(ys)$. Let $n \in Q_K(xs)$. Write $xn = E_n(xs)$ and $yn = E_n(ys)$. Since $n \in Q_K(xs)$, we have $xn \in Beh(K)$. On the other hand, yn is an infinite initial execution of L and $(xn, yn) \in F^\omega$. Flatness of F implies that yn is a behavior of L. This proves $n \in Q_L(ys)$. □

4. AN ETERNITY VARIABLE FOR REFINEMENT

We now develop an alternative for prophecy variables or backward simulations that is simpler and in a theoretical sense more powerful. Extending the metaphor of history and prophecy variables, they are named eternity variables, since they do not change during execution. They are simpler than prophecy variables in the sense that, below, both the proof of soundness in Theorem 4.1 and the proof of completeness in Theorem 4.4 are simpler than the corresponding proofs for prophecy variables. They are theoretically more powerful in the sense that their completeness does not require additional finiteness assumptions.

The idea is that an eternity variable has an indeterminate constant value, but that the states impose restrictions on this value. A behavior in which the eternity variable ever has a wrong value is simply discarded. Therefore, in every behavior, the eternity variable always has a value that satisfies all restrictions of the behavior.

The specification obtained by adding an eternity variable is called an eternity extension. In Section 4.1, we introduce eternity extensions, prove their soundness, and give a simple example. Completeness of eternity extension is proved in Section 4.2. At first sight, the use of eternity variables may seem to require arguing about complete behaviors rather than states and the next-state relation. As argued in Section 4.3, however, it is possible to combine the use of eternity variables conveniently with assertional methods.

4.1 Eternity Extensions Defined

Let K be a specification. Let M be a set of values for an eternity variable `m`. A binary relation R between $states(K)$ and M is called a *behavior restriction* of K iff, for every behavior xs of K, there exists an $m \in M$ with $(xs_i, m) \in R$ for all indices i:

$$\text{(BR)} \qquad xs \in Beh(K) \;\Rightarrow\; (\exists\, m :: (\forall\, i :: (xs_i, m) \in R)) \,.$$

If R is a behavior restriction of K, we define the corresponding *eternity extension* as the specification W given by

$$\begin{aligned}
&states(W) \;=\; R \,,\\
&start(W) \;=\; R \cap (start(K) \times M) \,,\\
&((x, m), (x', m')) \in step(W) \;\equiv\; (x, x') \in step(K) \;\wedge\; m = m' \,,\\
&ys \in prop(W) \;\equiv\; fst^\omega(ys) \in prop(K) \,.
\end{aligned}$$

It is clear that $step(W)$ is reflexive and that $prop(W)$ is a property. Therefore, W is a specification. It is easy to verify that $fst : states(W) \rightarrow states(K)$ is a refinement mapping. The soundness of eternity extensions is expressed by

THEOREM 4.1. *Let R be a behavior restriction. Then relation $cvf = cv(fst)$ is a flat simulation $K \rightarrowtriangle W$.*

PROOF. We first prove that *cvf* is a simulation. Let $xs \in Beh(K)$. We have to construct $ys \in Beh(W)$ with $(xs, ys) \in cvf^{\omega}$. By (BR), we can choose m with $(xs_i, m) \in R$ for all i. Then we define $ys_i = (xs_i, m)$. A trivial verification shows that the list ys constructed in this way is a behavior of W with $(xs, ys) \in cvf^{\omega}$. This proves that *cvf* is a simulation. Flatness of *cvf* follows directly from the definitions of flatness and $prop(W)$. □

The simulation $cvf : K \rightarrowtriangle W$ of Lemma 4.1 is called the eternity extension of K corresponding to behavior restriction R. In this construction, we fully exploit the ability to consider specifications that are not machine closed. Initial executions of W that cannot be extended to behaviors of W are simply discarded.

Remark 4.2. If M is a singleton set, such as the type `void`, the existential quantification in (BR) can be eliminated and condition (BR) reduces to the requirement that $D = \{x \mid (x, \_) \in R\}$ is an invariant. Then, W is isomorphic to the D-restricted specification K_D and *cvf* corresponds to the simulation $1_D : K \rightarrowtriangle K_D$ of Lemma 3.4(a) in 3.2.

Example 4.3. We give a simple example where a nontrivial eternity variable is used to prove that a given relation is a simulation. Let K be the specification given by the program

var j : *Nat* , b : *Boolean* ;
initially : j = 0 ∧ ¬b ;
do ¬b → j := j + 1 ;
[] j ≠ 0 → b := *true* ;
od ;
prop: b is sometimes *true*.

Let L be the specification given by

var k, n : *Nat* := 0, 0 ;
do n = 0 → k := 1 ; *choose* n ≥ 1 ;
[] k < n → k := k + 1 ;
od ;
prop: sometimes k = n.

Recall that the alternatives in the **do** loop are regarded as atomic. Let relation F between the state spaces of K and L be given by

$$((j, b), (k, n)) \in F \quad \equiv \quad j = k \ .$$

We claim that F is a simulation. In every behavior, specification L chooses the number of nontrivial steps of the behavior in the first nontrivial step. For K, this number is determined in the last nontrivial step. It thus needs prescience to construct the behavior of L from that of K.

We therefore factor relation F over an eternity extension. For this purpose, we form the eternity extension with eternity variable m : ℕ and relation

$$R : \quad \mathtt{j} \leq \mathtt{m} \quad \wedge \quad (\neg \mathtt{b} \quad \vee \quad \mathtt{j} = \mathtt{m}) \, .$$

The state of K remains constant once $\mathtt{b}$ has become *true*. Therefore, every behavior of K has a unique value for $\mathtt{m}$ that satisfies R, namely the final value of $\mathtt{j}$. This shows that R is a behavior restriction. We thus form the corresponding eternity extension $cvf : K \rightarrowtriangle W$. In view of behavior restriction R, specification W can be regarded as the program

var $\mathtt{j}, \mathtt{m} : Nat$, $\mathtt{b} : Boolean$;
initially: $\mathtt{j} = 0 \ \wedge \ \neg \mathtt{b}$;
do $\neg \mathtt{b} \ \wedge \ \mathtt{j} < \mathtt{m} \ \rightarrow \ \mathtt{j} := \mathtt{j} + 1$;
[] $\mathtt{j} = \mathtt{m} \neq 0 \ \rightarrow \ \mathtt{b} := true$;
od ;
prop: $\mathtt{b}$ is sometimes *true*.

Let $g : states(W) \rightarrow states(L)$ be given by

$$g(j, b, m) \ = \ (j, (j = 0 \,?\, 0 : m)) \,,$$

where $(\_?\_ : \_)$ stands for a conditional expression as in the language C. It is easy to see that g maps the initial states of W into the initial state of L. Every step in accordance with the first alternative of W is transformed into a step of L. Every step in accordance with the second alternative of W is transformed into a stuttering step of L. Every behavior of W is transformed into a behavior of L. Therefore, g is a refinement mapping $W \rightarrowtriangle L$. The composition $(cvf; g)$ is contained in relation F. This shows that F is a simulation. □

4.2 Completeness of Eternity Extensions

The combination of forward simulations, eternity extensions and refinement mappings is semantically complete in the following sense.

THEOREM 4.4. *Let $F : K \rightarrowtriangle L$ be a simulation that preserves quiescence. There exist a forward simulation $fw : K \rightarrowtriangle H$, an eternity extension $et : H \rightarrowtriangle W$ and a refinement mapping $g : W \rightarrowtriangle L$ such that $(fw; et; g) \subseteq F$.*

PROOF. In accordance with Lemma 3.5, the unfolding $cvl : K \rightarrowtriangle K^{\#}$ is a forward simulation. It therefore suffices to prove the following more specific result. □

LEMMA 4.5. *Let $F : K \rightarrowtriangle L$ be a simulation that preserves quiescence. The unfolding $cvl : K \rightarrowtriangle K^{\#}$ has an eternity extension $cvf : K^{\#} \rightarrowtriangle W$ and a refinement mapping $g : W \rightarrowtriangle L$ such that $(cvl; cvf; g) \subseteq F$.*

PROOF. We extend $K^{\#}$ with an eternity variable $\mathtt{m}$ in the set $Beh(L)$. For this purpose, let relation R between $states(K^{\#})$ and $Beh(L)$ consist of the pairs (xs, ys) such that, for some $xt \in Beh(K)$, it holds that

$$xs \sqsubseteq xt \ \wedge \ (xt, ys) \in F^{\omega} \ \wedge \ Q_K(xt) \subseteq Q_L(ys) \,.$$

We show that R is a behavior restriction by verifying condition (BR). Let uss be any behavior of $K^{\#}$. Define vss to be the stutterfree behavior of $K^{\#}$ with $vss \preceq uss$. By Lemma 3.6, we have that $xt = last^{\omega}(vss)$ is a behavior of K such that vss_i is a prefix of xt for all indices i. Since $F : K \rightarrowtriangle L$ preserves quiescence,

specification L has a behavior ys with $(xt, ys) \in F^\omega$ and $Q_K(xt) \subseteq Q_L(ys)$. This implies that $(vss_i, ys) \in R$ for all $i \in \mathbb{N}$. Since every element of uss is an element of vss, it follows that $(uss_i, ys) \in R$ for all $i \in \mathbb{N}$. Taking $\mathtt{m} = ys$, this proves condition (BR), so that R is a behavior restriction.

Let W be the R-eternity extension of $K^\#$. By Lemma 4.1, we have a simulation $cvf : K^\# \rightarrow\!\!\triangleright W$. Define $g : R \to states(L)$ by

$$g(xs, ys) = last(ys \mid \ell(xs)) .$$

We show that g is a refinement mapping from W to L. First, let $w \in start(W)$. Then w is of the form $w = (xs, ys)$ with $\ell(xs) = 1$. Therefore, $g(w) = last(ys|1) = ys_0 \in start(L)$. In every nonstuttering step in W, the length of xs is incremented with 1 and then we have $(ys_n, ys_{n+1}) \in step(L)$. Therefore, function g maps steps of W to steps of L.

In order to show that g maps every behavior of W to a behavior of L, it suffices to show that $g^\omega(ws) \in prop(L)$ for every stutterfree behavior of W. So, let ws be a stutterfree behavior of W. Since ws is a behavior of W, its elements have a common second component $ys \in Beh(L)$. We can therefore write $ws_k = (us_k, ys)$ for all k. Since $ws \in Beh(W)$, we have $us = fst^\omega(ws) \in Beh(K^\#)$. In particular, $(us_k, us_{k+1}) \in step(K^\#)$ for all k, and $last^\omega(us) \in prop(K)$.

We have $g(ws_k) = last(ys|\ell(us_k))$. Since ws is stutterfree, us is stutterfree. There are two possibilities. Either all elements of us are different or up to some index n all elements of us are different and from n onward they stay the same. This implies, that either $\ell(us_k) = k + 1$ for all k, or there exist a number n, such that $\ell(us_k) = \min(k, n) + 1$ for all k. In the first case, we have $g^\omega(ws) = ys \in prop(L)$. In the second case, $g^\omega(ws) = E_n(ys)$. Therefore, $g^\omega(ws) \in prop(L)$ would follow from $n \in Q_L(ys)$. Since $(us_n, ys) = ws_n \in R$, there exists a behavior ut of K such that $us_n \sqsubseteq ut$ and $(ut, ys) \in F^\omega$ and $Q_K(ut) \subseteq Q_L(ys)$. Since $\ell(us_n) = n + 1$, we have $us_n = (ut|n + 1)$. This implies that $E_n(ut)$ equals us_n followed by infinitely many states $ut_n = last(us_n)$. It follows that $E_n(ut) = last^\omega(us) \in prop(K)$ and hence $n \in Q_K(ut) \subseteq Q_L(ys)$.

It remains to prove $(cvl; cvf; g) \subseteq F$. Let (x, y) be in the lefthand relation. By the definition of $(cvl; cvf; g)$, there exist $xs \in states(K^\#)$ and $w \in states(W)$ with $x = last(xs)$ and $xs = fst(w)$ and $g(w) = y$. By the definition of W, we can choose $ys \in Beh(L)$ with $w = (xs, ys)$. Let $n = \ell(xs)$. Then $x = xs_{n-1}$ and $y = g(w) = ys_{n-1}$. Since $(xs, ys) \in R$, we also have $(x, y) = (xs_{n-1}, ys_{n-1}) \in F$. This proves the inclusion. □

Remark 4.6. Theorem 4.4 is more relevant than Lemma 4.5 since it suggests the flexibility to add conveniently many history variables, and not more than necessary.

The converse of Theorem 4.4 also holds. In fact, forward simulations, eternity extensions and refinement mappings are flat simulations, which preserve quiescence by Lemma 3.12. Since preservation of quiescence is compositional, it follows that every simulation F that satisfies the consequent of Theorem 4.4 preserves quiescence.

4.3 Behavioral or Assertional Reasoning?

In general, there are two methods for the verification of concurrent algorithms (as discussed, for example, in Hesselink [2002a], p. 344). One method, the assertional approach, is to rely on invariants and variant functions. The alternative, the behavioral approach, is to argue about execution sequences (behaviors) where certain actions precede other actions. We prefer the assertional approach (see also Hesselink [1998] where we described it as the synchronic approach). Yet, it is clear that, in the analysis of an algorithm that gradually modifies the state, we cannot avoid temporal or behavioral arguments completely. We therefore strive at a separation of concerns where the behavioral argument is a formal triviality and all complexity of the algorithm is treated at the level of states and the next-state relation.

One may object that our proof obligation (BR) in 4.1 requires quantification over all possible behaviors, which is precisely what the assertional methods try to avoid. This objection is not justified. In fact, it could be raised equally well against the use of invariants, defined as predicates that hold in all reachable states.

The question thus boils down to establishing condition (BR) of 4.1. Given a behavior xs, one has to construct a value m for the eternity variable such that $(\forall\ i\ ::\ (xs_i, m) \in R)$. In practice, we proceed as follows: First, rephrase $(\forall\ i\ ::\ (xs_i, m) \in R)$ as $(\forall\ i\ ::\ xs_i \in R(m))$ for a state predicate $R(m)$, with a free variable m yet to be determined. Predicate $R(m)$ plays the same role as an invariant, but only for a specific behavior xs.

We now use that Theorem 4.4 allows us the introduction of history variables. We introduce a history variable the value of which converges in a certain sense for every behavior, and we use the "limit" as a value for m. In Example 3.10, the final value of the variable `j` was this limit.

In our more interesting examples (see Section 5 and Hesselink [2004]), the eternity variable m is an infinite sequence and the approximating history variable consists of a pair `(n, a)` where `n` holds a natural number and `a` is an infinite array filled upto `n`. This pair is modified only by steps of the form

$$\langle\ \mathtt{a[n]} := \mathit{expression}\ ;\ \mathtt{n} := \mathtt{n} + 1\ \rightarrow\ .$$

The behavior restriction is given as the state predicate

$$(*) \qquad R(m)\ \equiv\ (\forall\ j : j < \mathtt{n} : m(j) = \mathtt{a}[j])\ .$$

Since `n` is incremented only and `a` is never modified at indices below `n`, for every behavior xs, the existence of a value m that always satisfies $R(m)$ is a formal triviality.

In our applications, this is the only behavioral argument needed. The remainder of the verification can be done by assertional methods. Of course, creativity is needed to come up with approximating history variables that carry enough information, but this is the same kind of creativity as needed to invent invariants.

When we restrict the method to behavior restrictions of the special kind (*), we cannot maintain completeness, since in the proof of Lemma 4.5 we used a

different kind of behavior restriction. So, indeed, we cannot guarantee that in all applications there is a convenient reduction to the assertional setting.

5. A SLIGHTLY BIGGER EXAMPLE

In this section, we illustrate the theory by a tiny application. We prove that a relation between the state spaces of specifications *K0* and *K1* is a simulation by factoring it over the forward simulation, an eternity extension, an invariant restriction, and two refinement mappings.

5.1 The Problem

Let *K0* be the specification corresponding to the program

var $\mathtt{j} : Nat := 0$;
do *true* $\rightarrow$ $\mathtt{j} := \mathtt{j} + 1$
[] $\mathtt{j} > 0 \rightarrow \mathtt{j} := 0$
od ;
prop: j decreases infinitely often.

The fairness assumption requires that the second alternative is chosen infinitely often. Specification *K0* has $states(K0) = \mathbb{N}$ and $start(K0) = \{0\}$ and relation $step(K0)$ given by

$$(j, j') \in step(K0) \quad \equiv \quad j' \in \{0, j, j+1\} \; .$$

The fairness assumption is expressed in $prop(K0) = \Box\Diamond[\![>]\!]$.

We extend specification *K0* with a variable z that guesses when j will jump back. We thus obtain the extended specification *K1* with the program

var $\mathtt{j}, \mathtt{z} : Nat := 0, 0$;
do $\mathtt{j} < \mathtt{z} \rightarrow \mathtt{j} := \mathtt{j} + 1$;
[] $\mathtt{j} = 0 \rightarrow \mathtt{j} := 1$; *choose* $\mathtt{z} \geq 1$;
[] $\mathtt{j} = \mathtt{z} \rightarrow \mathtt{j} := 0$; $\mathtt{z} := 0$;
od ;
prop: (j, z) changes infinitely often.

Recall that the alternatives in the loop are executed atomically. The supplementary property only ensures that behaviors do not stutter indefinitely. We thus have $prop(K1) = \Box\Diamond[\![\neq]\!]$.

The function $f_{1,0} : states(K1) \rightarrow states(K0)$ given by $f_{1,0}(j, z) = j$ is easily seen to be a refinement mapping $K1 \rightarrowtriangle K0$.

More interesting is the converse relation $F_{0,1} = cv(f_{1,0})$. It is not difficult to show by ad-hoc methods that $F_{0,1}$ is a simulation $K0 \rightarrowtriangle K1$, but the aim of this section is to do it systematically by means of the theory developed.

In comparison with *K0*, the variable z seems to prophecy the future behavior. This suggests to use a backward simulation. Our best guess is the relation $F = \{(j, (j, z)) \mid j \leq z\}$ between the state spaces of *K0* and *K1*. Indeed, relation F satisfies three of the four conditions for backward simulations, but condition (B2) fails: the sets $\{y \mid (x, y) \in F\}$ are always infinite. We therefore use factorization over an eternity extension.

5.2 A History Extension to Approximate Eternity

Every behavior of *K1* contains infinitely many steps where a new value for z is chosen. These values are prophecies with respect to *K0*. In the behaviors of *K0*, these values can only be seen at the jumping steps. We therefore extend *K0* with an infinite array of history variables to record the subsequent jumping values.

We thus extend specification *K0* with two history variables n and q. Variable n counts the number of backjumps of j, while q is an array that records the values from where j jumped.

var j : *Nat* := 0 , n : *Nat* := 0 ,
  q : **array** *Nat* **of** *Nat* := ([*Nat*] 0) ;
do *true* → j := j + 1 ;
[] j > 0 → q[n] := j ; n := n + 1 ; j := 0 ;
od ;
prop: j decreases infinitely often.

This yields a specification *K2* with the supplementary property $\Box\Diamond[\![\, \mathtt{j} > \mathtt{j}' \,]\!]$ where $\mathtt{j}'$ stands for the value of j in the next state.

It is easy to verify that the function $f_{2,0} : \mathit{states}(K2) \to \mathit{states}(K0)$ given by $f_{2,0}(j, n, q) = j$ is a refinement mapping. Its converse $F_{0,2} = \mathit{cv}(f_{2,0})$ is a forward simulation $K0 \rightarrowtriangle K2$. Indeed, the conditions (F0) and (F2) hold almost trivially. As for (F1), if we have related states in *K0* and *K2*, and the state in *K0* makes a step, it is clear that *K2* can take a step such that the states remain related. The variables n and q are called history variables since they record the history of the execution.

5.3 An Example of an Eternity Extension

We now extend *K2* with an eternity variable m, which is an infinite array of natural numbers with the behavior restriction

$$R: \quad (\forall\, i : 0 \le i < \mathtt{n} : \mathtt{m}[i] = \mathtt{q}[i]) \,.$$

We have to verify that every behavior of *K2* allows a value for m that satisfies condition R. So, let *xs* be an arbitrary behavior of *K2*. Since j jumps back infinitely often in *xs*, the value of n tends to infinity. This implies that $\mathtt{q}[i]$ is eventually constant for every index i. We can therefore define function $m : \mathbb{N} \to \mathbb{N}$ by $\Diamond\Box[\![\, m(i) = \mathtt{q}[i] \,]\!]$ for all $i \in \mathbb{N}$. It follows that $\Box[\![\, i < \mathtt{n} : \Rightarrow : m(i) = \mathtt{q}[i] \,]\!]$ for all i. This proves that m is a value for m that satisfies R for behavior *xs*.

Let *K3* be the resulting eternity extension and $F_{2,3} : K2 \rightarrowtriangle K3$ be the simulation induced by Lemma 4.1. Specification *K3* corresponds to the program

var j : *Nat* := 0 , n : *Nat* := 0 ,
  q : **array** *Nat* **of** *Nat* := ([*Nat*] 0) ,
  m : **array** *Nat* **of** *Nat* {*arbitrary*} ;
do *true* → j := j + 1 ;
[] j = m[n] > 0 → q[n] := j ; n := n + 1 ; j := 0 ;
od ;
prop: j decreases infinitely often.

5.4 Using Refinement Mappings and an Invariant

We first eliminate array q, which has played its role. This gives a refinement mapping $f_{3,4}$ from *K3* to the specification *K4* with program

var j : *Nat* := 0 , n : *Nat* := 0 ,
  m : **array** *Nat* **of** *Nat* {*arbitrary*} ;
do *true* → j := j + 1 ;
[] j = m[n] > 0 → n := n + 1 ; j := 0 ;
od ;
prop: j decreases infinitely often.

Since j must decrease infinitely often in *K4*, the occurring states of *K4* satisfy the invariant

$$D: \quad \mathtt{j} \le \mathtt{m}[\mathtt{n}] \ \wedge \ (\forall\, i :: \mathtt{m}[i] \ge 1)\,.$$

Note that D is not a forward invariant of *K4* (see Section 2.1). Let *K5* be the D-restriction of *K4*, with the simulation $1_D : K4 \rightarrowtriangle K5$ of Lemma 3.4(a). Specification *K5* corresponds to

var j : *Nat* := 0 , n : *Nat* := 0 ,
  m : **array** *Nat* **of** *Nat with* (∀ *i* :: m[*i*] ≥ 1) ;
do j < m[n] → j := j + 1 ;
[] j = m[n] > 0 → n := n + 1 ; j := 0 ;
od ;
prop: j decreases infinitely often.

Let function $f_{5,1} : states(K5) \to states(K1)$ be defined by

$$f_{5,1}(j,n,m) \quad = \quad (j, (j = 0\,?\,0 : m[n]))\,,$$

again using a C-like conditional expression. We verify that $f_{5,1}$ is a refinement mapping. Since $f_{5,1}(0,0,m) = (0,0)$, initial states are mapped to initial states. We now show that a step of *K5* is mapped to a step of *K1*. By convention, this holds for a stuttering step. A nonstuttering step that starts with $\mathtt{j} = 0$ increments j to 1. The $f_{5,1}$-images make a step from $(0,0)$ to $(1,r)$ for some positive number z. This is in accordance with *K1*. A step of *K5* that increments a positive j has the precondition $\mathtt{j} < \mathtt{m}[\mathtt{n}]$; therefore, the $f_{5,1}$-images make a *K1*-step. A back-jumping step of *K5* has precondition $\mathtt{j} = \mathtt{m}[\mathtt{n}] > 0$. Again, the $f_{5,1}$-images make a *K1*-step. It is easy to see that $f_{5,1}$ transforms behaviors of *K5* to behaviors of *K1*.

We thus have a composed simulation $G = (F_{0,2}; F_{2,3}; f_{3,4}; 1_D; f_{5,1}) : K0 \rightarrowtriangle K1$. One can verify that $(j,(k,m)) \in G$ implies $j = k$. It follows that the above relation $F_{0,1}$ satisfies $G \subseteq F_{0,1}$. Therefore, $F_{0,1}$ is a simulation $K0 \rightarrowtriangle K1$. This shows that an eternity extension can be used to prove that $F_{0,1}$ is a simulation $K0 \rightarrowtriangle K1$.

Remark 5.1. We have taken more steps here than accounted for in Theorem 4.4. By taking a different behavior restriction R, we could have compressed the last three steps into one more complicated step.

6. CONCLUSIONS AND FUTURE WORK

We have introduced simulations of specifications to unify all cases where an implementation relation can be established. This unifies refinement mappings, history variables or forward simulations, and prophecy variables or backward simulations, and refinement of atomicity as in Lipton's Theorem [Cohen and Lamport 1998; Lipton 1975]. This unification is no great accomplishment: a general term to unify distinct kinds of extensions is useful for the understanding, but methodologically void.

We have introduced eternity extensions as variations of prophecy variables and backward simulations. We have proved semantic completeness: every simulation that preserves quiescence can be factored as a composition of a forward simulation, an eternity extension and a refinement mapping. The restrictive assumptions machine-closedness and finite invisible nondeterminism, as needed for completeness of prophecy variables or forward-backward simulations in Abadi and Lamport [1991] and Lynch and Vaandrager [1995] are superfluous when eternity variables are allowed. The assumption of internal continuity is weakened to preservation of quiescence.

The theory has two versions. In the strict version presented here, we allow the concrete behaviors to take more but not less computation steps than the abstract behaviors. This is done by allowing additional stutterings to the abstract specifications. The strict theory is also the simpler one and it results in a finer hierarchy of specifications than the stuttering theory.

It is likely that the results of this article can be transferred to input-output automata and labeled transition systems. The ideas may also be useful in specifications and correctness arguments for real-time systems.

As indicated above, we developed the theory of eternity variables to apply them in Hesselink [2004] to the serializable database interface problem of Broy [1992], Lamport [1992] and Schneider [1992]. The practicality of the use of eternity variables is witnessed by the fact that the proof in Hesselink [2004] is verified by means of the mechanical theorem prover NQTHM Boyer and Moore [1997], which is based on first-order logic.

ACKNOWLEDGMENTS

I am grateful to Eerke Boiten, Leslie Lamport, Caroll Morgan, and Gerard Renardel de Lavalette for encouragements, comments, and profound discussions.

REFERENCES

ABADI, M. AND LAMPORT, L. 1991. The existence of refinement mappings. *Theoret. Comput. Sci. 82*, 253–284.

ABADI, M. AND LAMPORT, L. 1995. Conjoining specifications. *ACM Trans. Program. Lang. Syst. 17*, 507–534.

BOYER, R. S. AND MOORE, J S. 1997. *A Computational Logic Handbook*. Academic Press, Orlando, Fla.

BROY, M. 1992. Algebraic and functional specification of an interactive serializable database interface. *Distr. Comput. 6*, 5–18.

CLINT, M. 1973. Program proving: coroutines. *Acta Inf. 2*, 50–63.

COHEN, E. AND LAMPORT, L. 1998. Reduction in TLA. In *CONCUR '98*, D. Sangiorgi and R. de Simone, Eds. Lecture Notes in Computer Science, vol. 1466. Springer, New York, 317–331.

DE ROEVER, W.-P., DE BOER, F., HANNEMANN, U., HOOMAN, J., LAKHNECH, Y., POEL, M., AND ZWIERS, J. 2001. *Concurrency Verification, Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, Cambridge.

EILENBERG, S. AND MACLANE, S. 1945. General theory of natural equivalences. *Trans. AMS 58*, 231–294.

HE, J., HOARE, C. A. R., AND SANDERS, J. W. 1986. Data refinement refined. In *ESOP 86*, B. Robinet and R. Wilhelm, Eds. Lecture Notes in Computer Science, vol. 213. Springer, New York, 187–196.

HESSELINK, W. H. 1998. Invariants for the construction of a handshake register. *Inf. Proc. Lett. 68*, 173–177.

HESSELINK, W. H. 2002a. An assertional criterion for atomicity. *Acta Inf. 38*, 343–366.

HESSELINK, W. H. 2002b. Eternity variables to simulate specifications. In *MPC 2002*, E. Boiten and B. Moeller, Eds. Lecture Notes in Computer Science, vol. 2386. Springer, New York, 117–130.

HESSELINK, W. H. 2004. Using eternity variables to specify and prove a serializable database interface. *Sci. Comput. Program. 51*, (July), 47–85.

HESSELINK, W. H. 2005. Universal extensions to simulate specifications. http.//www.cs.rng.nl/~wim/pub/mans.html.

JONNSON, B. 1991. Simulations between specifications of distributed systems. In *CONCUR '91*, J. Baeten and J. Groote, Eds. Lecture Notes in Computer Science, vol. 527. Springer, New York, 346–360.

JONSSON, B., PNUELI, A., AND RUMP, C. 1999. Proving refinement using transduction. *Distrib. Comput. 12*, 129–149.

LADKIN, P., LAMPORT, L., OLIVIER, B., AND ROEGEL, D. 1999. Lazy caching in TLA. *Distrib. Comput. 12*, 151–174.

LAMPORT, L. 1989. A simple approach to specifying concurrent systems. *Commun. ACM 32*, 32–45.

LAMPORT, L. 1992. Critique of the Lake Arrowhead three. *Distrib. Comput. 6*, 65–71.

LAMPORT, L. 1994. The temporal logic of actions. *ACM Trans. Prog. Lang. Syst. 16*, 872–923.

LIPTON, R. J. 1975. Reduction: a method of proving properties of parallel programs. *Commun. ACM 18*, 717–721.

LYNCH, N. AND VAANDRAGER, F. 1995. Forward and backward simulations. Part I: Untimed systems. *Inf. Comput. 121*, 214–233.

MANNA, Z. AND PNUELI, A. 1995. *Temporal Verification of Reactive Systems: Safety*. Springer, New York.

MILNER, R. 1971. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*. British Comput. Society, 481–489.

OWICKI, S. AND GRIES, D. 1976. An axiomatic proof technique for parallel programs. *Acta Inf. 6*, 319–340.

SCHNEIDER, F. B. 1992. Introduction. *Distrib. Comput. 6*, 1–3.

Received August 2002; revised August 2003; accepted August 2003